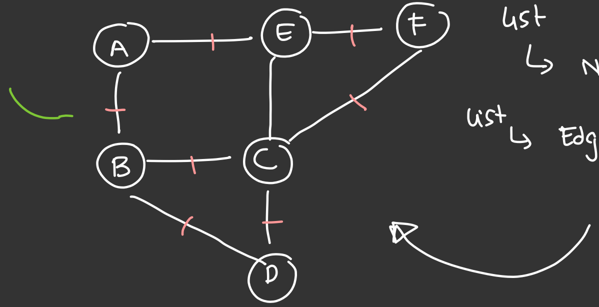
" Graphs - I "

→ **Data Structure** - network of nodes and edges

6 N, 7 E

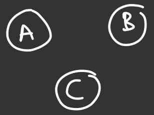
$\boxed{\text{Edge list}}$



List ↳ Nodes = {A, E, F, B, C, D}

List ↳ Edges = {
AE,
EF,
FC,
BC,
AB,
BD,
CD, EC
}

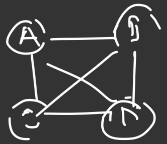
Min Edges - 0



Max Edges - Every pair of Nodes

$^{N}C_2 \propto N^2$

$^{4}C_2 = 6$

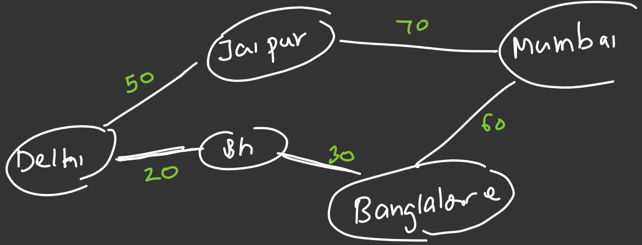


Real life use

Landmarks → Nodes/vertices

Roads → Edges

Cost → Length/
Travel time



Edge list

(Delhi, Jaipur, 50)

(Delhi, Bhopal, 20)

⋮

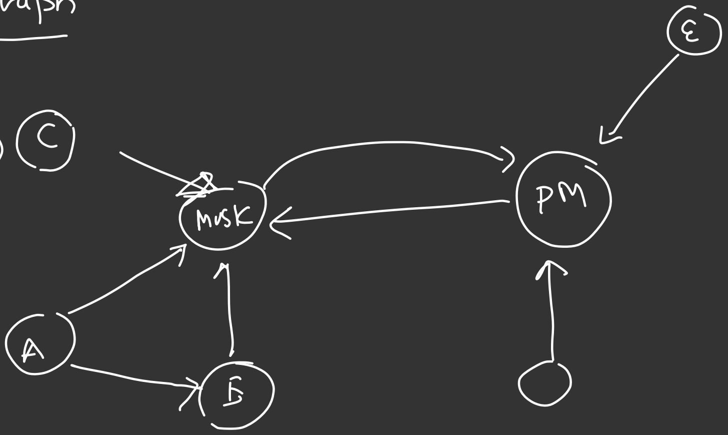
Undirected weighted graph

↓ egdes don't have a specific direction (2-way-road)

↓ each edge has wt /cost

"Directed" graph

Twitter (directed)



Instagram (directed)

A → B

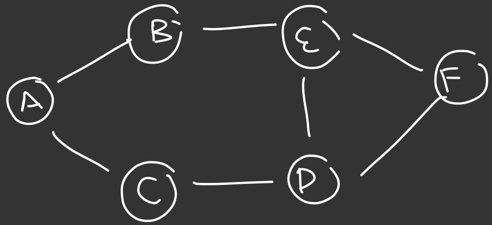
~~B → A~~

Facebook (undirected)
unweighted

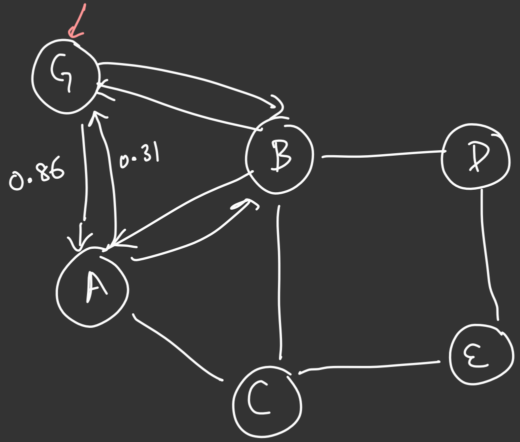
↓

friend

A ⟶ B
⟵



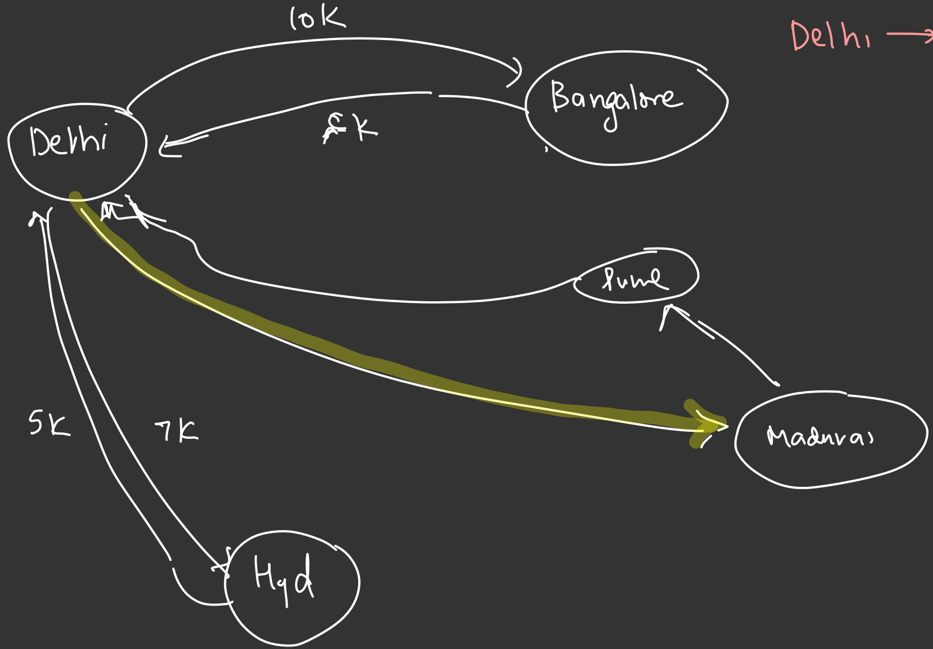
- Facebook graph (directed , weighted graph as well))



A −G	0·31
G − A	0·86

⋮

Flights "weighted", "Directed"

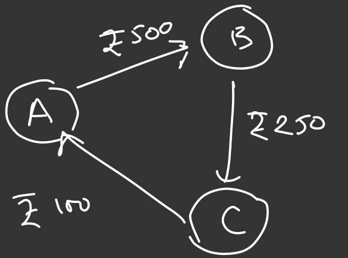


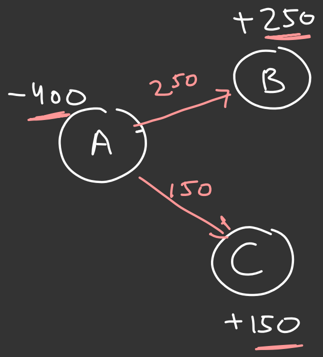
Delhi → Madmai

More examples

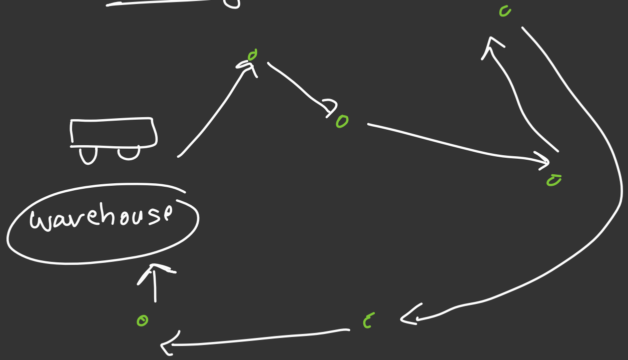
① Splitwise

Directed & weighted





② Delivery optimisation



min cost of the Tour (Travelling Salesman Problem) → Bitmasking Class

PCB Design

↓

Min Copper wire so that all components are connected

③ Google Maps,

friendsuggestions

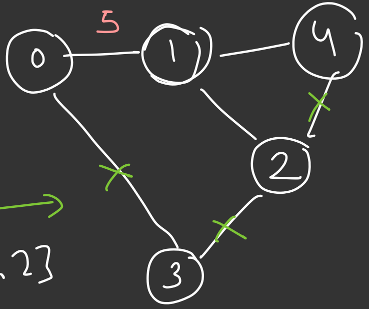
⋮

☆ important from Interview Perspective

Create / Store graph

① Edge list

List<int> Nodes = {0, 1, 2, 3, 4}



List < Edge > Nodes = { {0,1} {1,4} {1,2}

---- }

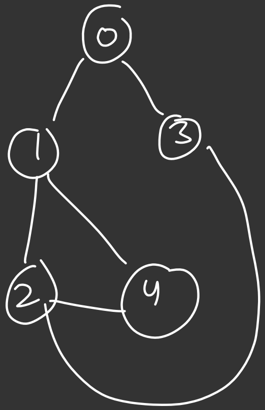
→ (useful in certain cases – where sorting of edges is reqd)

Disadv

Find nbrs of 1 ⇒ O(E)

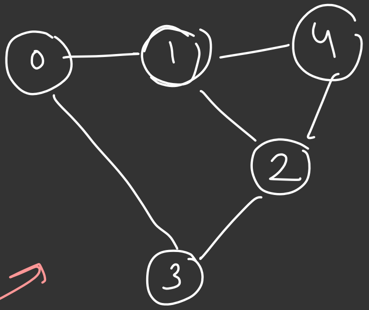
E: Edges

② Adj Matrix



	0	1	2	3	4
0	0	✓	0	✓	0
1	✓	0	✓	0	✓
2	0	✓	0	✓	✓
3	✓	0	✓	0	0
4	0	✓	✓	0	0

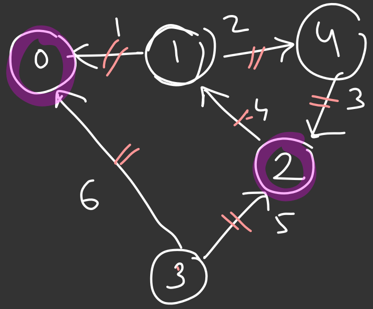
bool



	0	1	2	3	4
0	∞	∞	∞	∞	∞
1	1	∞	∞	∞	2
2	∞	4	∞	∞	∞
3	6	∞	5	∞	∞
4	∞	∞	3	∞	∞

mat

~~bool~~ int



✓ - dir

✓ - undir

✓ - wt

✓ - unwt

10^8 cells

Adv ⇓

⇒ O(1) lookup

for an

edge Node x-y

Do we have edge x-y ?

mat[x][y]

Disadv ⇑

⇒ $O(N^2)$ Space + time

⇒ Matrix is thinly populated

$E << N^2$ → Space wastage

⇒ Find nbrs of $\boxed{X}$ → O(N) ⇒ expensive

N= 10^5

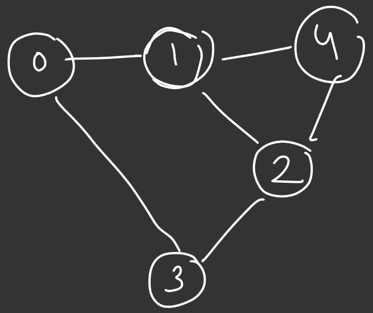
10^{10} → huge

↓

$\boxed{\text{OOM}}$

③ Adjacency List

Maintain a list for each node to store the nbrs of that node



N = 5

adjList

- 0 → 1,3 ✓ l[0]
- 1 → 0,4,2 ✓ l[1].
- 2 → 1,3,4 ✓ l[2]
- 3 → 0,2 ✓ l[3]
- 4 → 1,2 ✓ l[4]

Array of lists

① list < int > l[5]

linkedlist or ArrayList

Input

⇒ 0 - 1

⇒ 0 - 3

⇒ 1 - 4

⇒ 1 - 2

⇒ 2 - 3

⇒ 2 - 4

an array where bucket of the array holds a list obj

Input   x - y

for every edge in Input => { l[x]. add(y)
l[y]. add(x)

add -> Inbuitt method in list class

list[0]

0 → (1,5) (3,3)

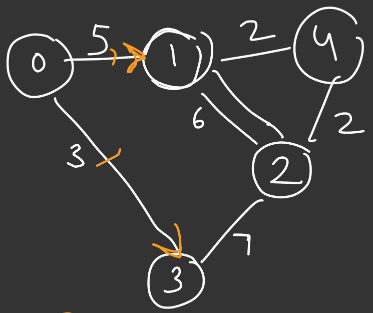
1 → (2,6) - -

2 → - - -

3 → - - -

4 → - - -

list< pair<int, int> >



Input   x - y - wt

0 - 1 - 5

0 - 3 - 3

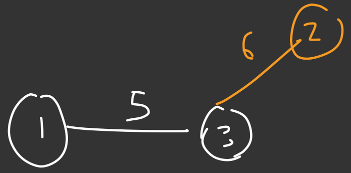
lisl[0] add (pair(1,5))

list[0] add (pair(3,3))

weighted graph ✓

```
x - y - wt

{
l[x] add ( pair(y, wt))
 if(undir) {
       l[y].add ( pair(x, wt))
 }
}
```



1 → (3,5)

3 → (1,5) (2,6)

2 → (3,6)

list of cities who are nbrs

city

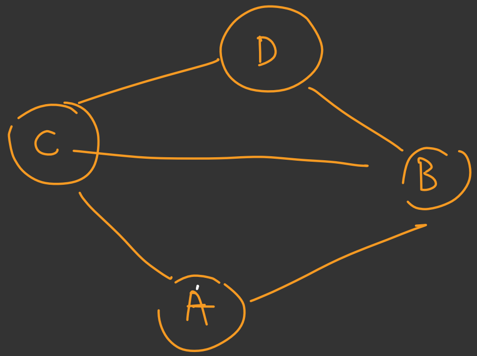
A → (B, C)

C → (A, B, D)

Key ← Value

non-numeric data in graph

hashmap < String, list<String> >   hm



hm["A"]. add ("B")

A — B

x - y →

hm[x] add (y)

✓ Advantages . ① Space efficient

Space = $O(V+E)$

☆ ☆ ☆ ② Directly itreate on nbrs of a node

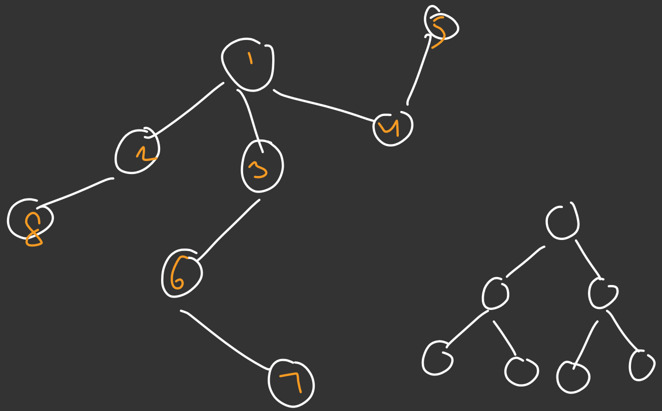
✓ Disadv - No $O(1)$ lookup for an edge b/w $x-y$

↓

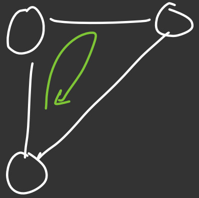
not really reqd !

Tree vs graph

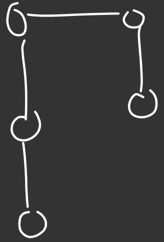
14-Dec
(wed.)



- 8 Nodes N
- exactly 7 edges N-1 edges
- tree is a graph but without any cycle



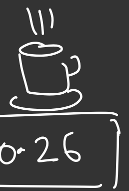
graph
(cyclic)



graph
(non-cyclic)

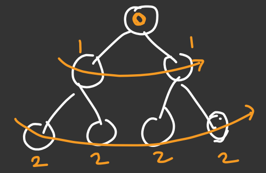
edges → 0 to N^2

~~Every graph is a Tree~~



BFS / Breadth First Search

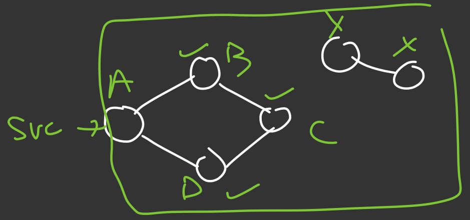
↳ Traversal all nodes of the graph starting from any source node



Use

✓→ check if there is path from "src" to "dest"

✓→ if it is unweighted, can give the shortest path to all the nodes reachable from source

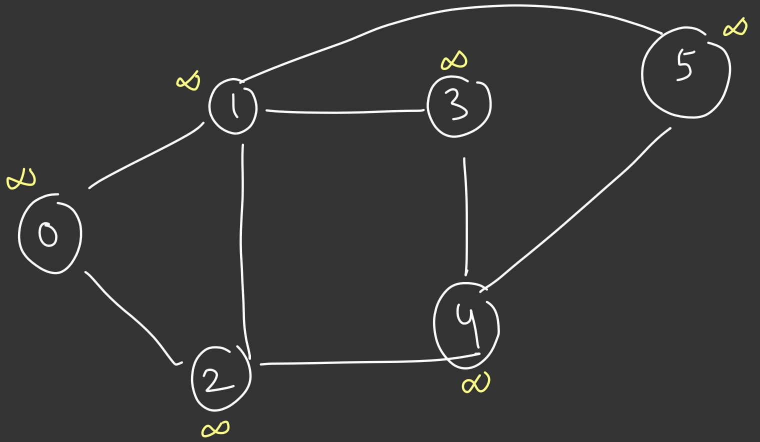


[Single Source Shortest Path]

↓

in a unweighted graph

A → B
→ C
→ D }



N = 6

Src = 0

Shortest "path len" to all other nodes

⇒ Need to track whether or not a node has been visited earlier

bool visited[N]

int dist[N]

Adj List

0 → 1, 2

1 → 0, 2, 3, 5

2 → 0, 1, 4

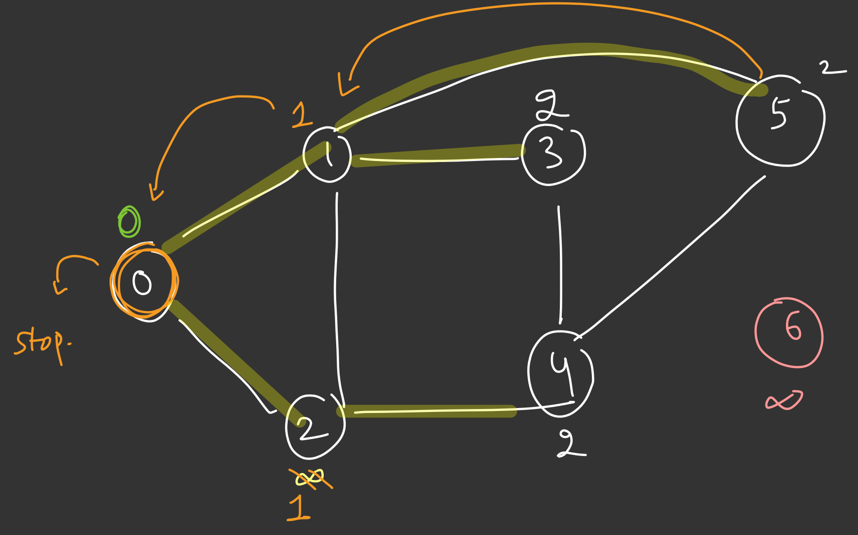
3 → 1, 4

4 → 2, 3, 5

5 → 1, 4

dist

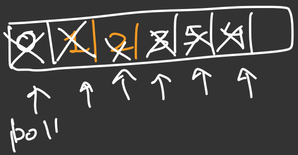
∞	∞	∞	∞	∞	∞
0	1	2	3	4	5



```
queue<int> q

dist[src] = 0

q.add(src)
```



```
while(!q.empty()){

    f = q.poll()

    for(every nbr of f){
        if(dist[nbr] == ∞){
            parent[nbr] = f
            dist[nbr] = 1 + dist[f]

            q.add(nbr)
        }
    }
}
```

Adj List

⇒ 0 → 1, 2

⇒ 1 → ~~0~~, ~~2~~, 3, 5

⇒ 2 → ~~0~~, ~~1~~, ~~4~~

⇒ 3 → 1, 4

⇒ 4 → 2, 3, 5

⇒ 5 → 1, 4

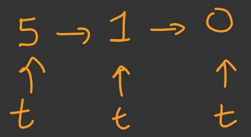
dist
0 → 0
1 → 1
2 → 1
3 → 2
4 → 2
5 → 2
6 → ∞

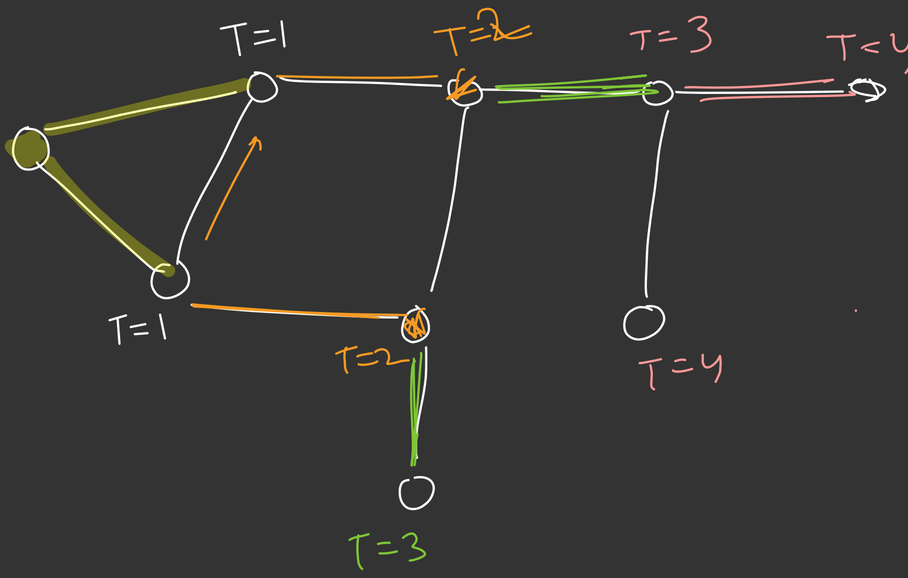
parent
0 → -1
1 → 0
2 → 0
3 → 1
4 → 2
5 → 1

" Route from src to dest "

Tracing the Shortest Path →

```
temp = dest
while( parent[temp] != -1 ) {
    print(temp)
    temp = parent[temp]
}
```



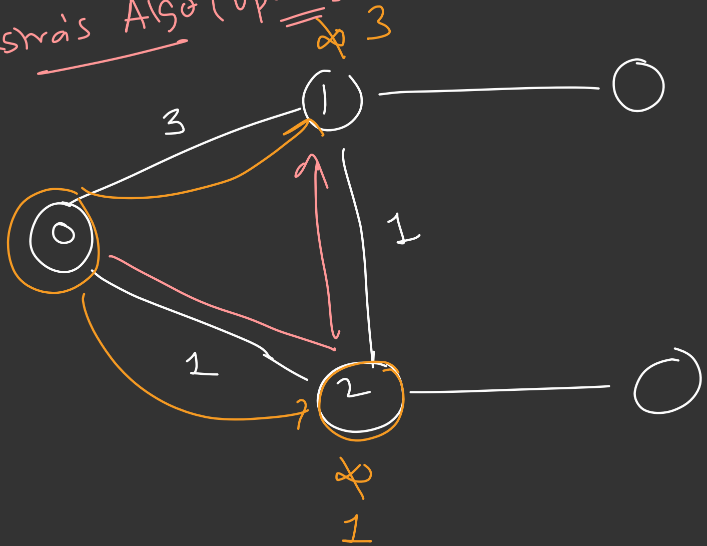


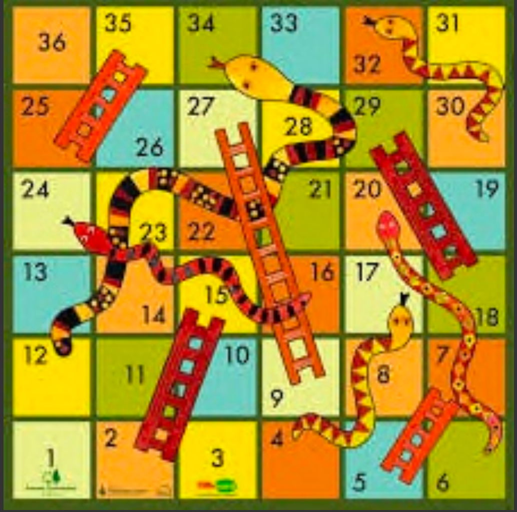
Weighted graph → ~~BFS~~

Dijkstra's Algo (upcoming)

dist[1] = ~~3~~, 2

dist[2] = 1





Snake Ladder

start - 0

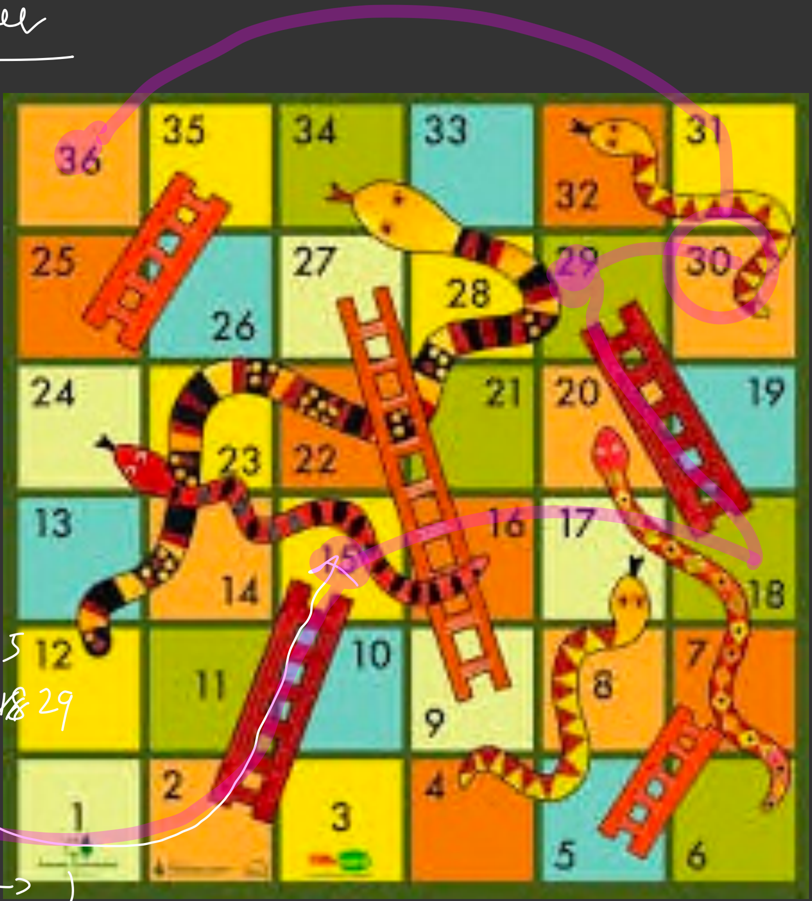
Throw anything

1
2
3
4
5
6

dice throws.
↑
Min Moves need to reach 36

- climb ladder bottom
- cut by Snake head
- Can't go backward using dice throw

Throw 2-15
Thr -3 - ~~18~~ 29
Th -1 - 30
— 6-36



0 → 1
0 — 2 → 15
0 — 3
0 — 4

1 → 2
1 → 3
1 → 4

Win ⇒ 4 moves

⇒ graph

1 → 5
1 → 6
1 → 7

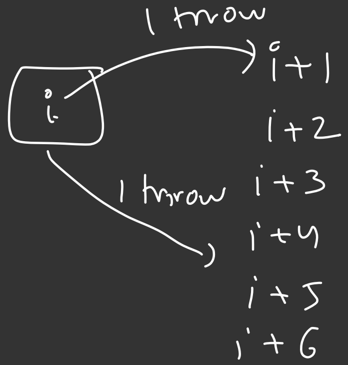
Nodes → 36

Edges (apprx) - 6 (Nodes) + Snakes + ladders

↑ input ↑

loop.

1 → 5
5 → 1



Directed → ✓

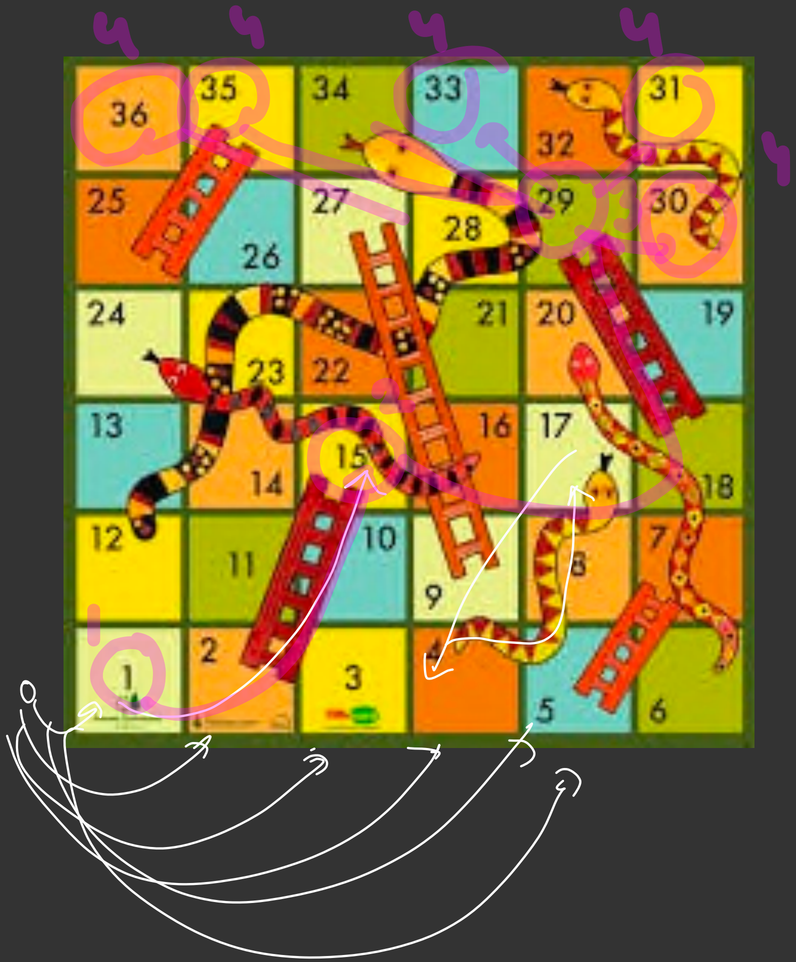
Weighted → ✗

✓ Directed unweighted graph

↑

BFS S.S.S.P

⇒ Each dice throw is 1 cost (1 move)



Create

board[36] = {0}

$$= \left[\begin{array}{cccccccc} \_\_, & 0, & 13, & \_\_, & \_\_, & -13 & - - - - & \\ 0 & 1 & 2 & 3 & 4 & 17 & & 36 \end{array} \right]$$

✓ Snakes

4 - 17 = -13

⋮

i - j

board[j] = i-j

✓ Ladder

2-15 = +13

9-27 = +25

⋮

board[i] = j-1

```
graph g(36)

graph  for (i=0;  i<36 ; i++){
           for(dice =1 ; dice <=6; dice++){
               pos = i+dice
               pos = pos + bord[pos]
               if(pos ≤ 36){
                   g.addEdge(i, pos)
           }       }
}

g.bfs(0,36)
            ↑
```

5 → 6
7
8
9
$\boxed{10}$ → 8
11

i=1   dice =1

pos = 1+1 + bord[2]

= 2+13

= $\boxed{15}$